

# Path Planning

For the second part of the project, we switched to Python for the path planning because it offers more useful libraries. We used the A\* algorithm and allowed 18 different types of moves: the four cardinal directions (up, down, left, right), the four diagonals, pure rotation moves, and combinations of rotation plus movement. We skipped diagonal-plus-rotation to keep things simpler.

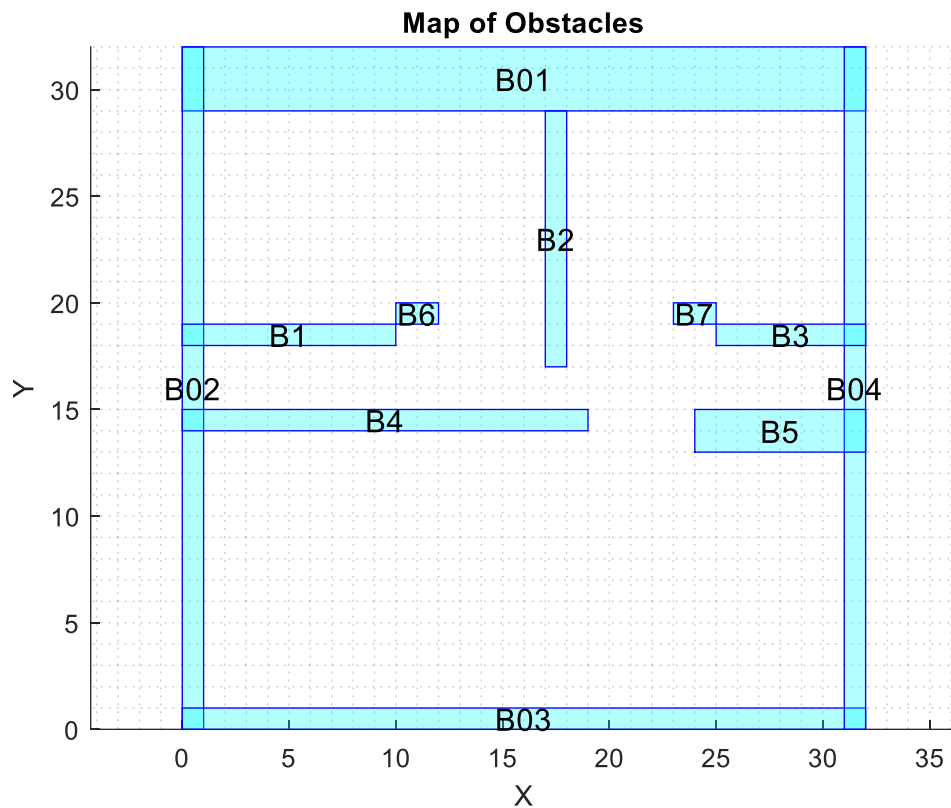
We measured distances between nodes with the Euclidean formula:

$$d = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

And set the cost of a move to be that distance plus a small extra cost for rotation. This way, the planner prefers straight lines when possible.

$$Cost = \sqrt{d_x^2 + d_y^2} + 0.1 \cdot \theta$$

The original grid given was  $32 \times 30$ , but both parts of the project needed a  $32 \times 32$  grid. To make it fit, we slightly extended obstacle B01 to fill the missing space.



We ran into two main problems. The first was that the grid resolution was too low, if even a tiny part of a cell touched an obstacle in the C-space, the whole cell got marked as blocked. That made

paths unnecessarily tight or impossible. We fixed it by doubling the resolution to  $64 \times 64$  reducing the cell size to 0.5 units.

The second was  $\theta$  resolution. Our robot is an  $8 \times 1$  rectangle rotating about its bottom left corner. The farthest vertex from the pivot is:

$$r = \sqrt{8^2 + 1^2} = \sqrt{65} \approx 8.0622$$

The maximum displacement of that vertex for a single change in  $\theta$  is:

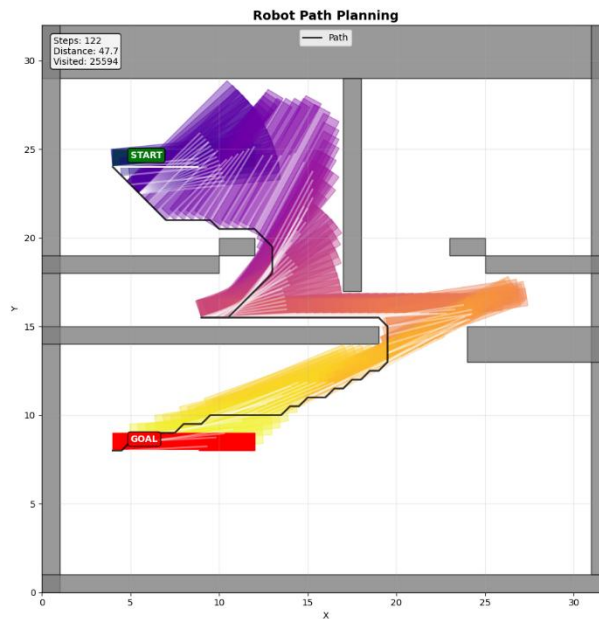
$$\frac{2\pi}{N_{layers}} \cdot r = \frac{16.1244\pi}{N_{layers}}$$

To avoid “skipping” over cells, this displacement must be less than the cell size (0.5 for our grid). That gives:

$$\frac{16.1244\pi}{N_{layers}} \leq 0.5 \rightarrow N_{layers} \geq 101.31$$

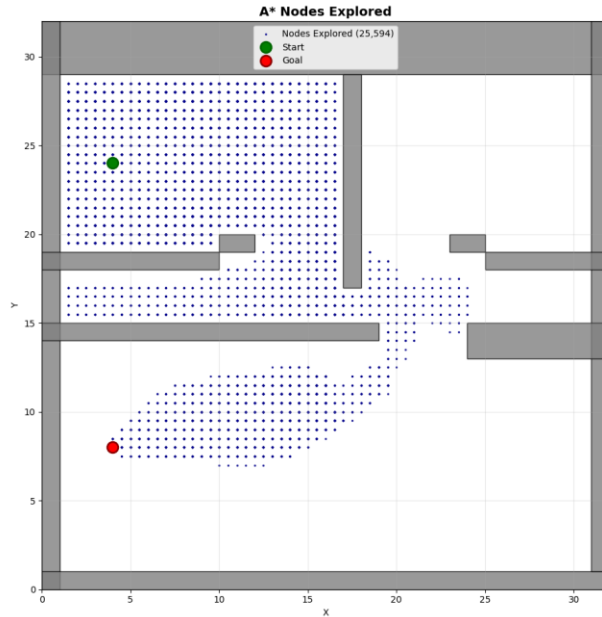
We chose 128 layers to ensure the rotation between consecutive  $\theta$  slices were small enough.

Putting everything together we get:



As we can see, no part of the robot touches the borders, and it gets from the start to the goal while rotating. The total number of steps taken was 122, the total distance traveled was 47.7 and the number of nodes needed to get this solution was 25594.

When we plot the explored nodes, we get stretched oval shapes along the path. Obstacles break them up, so instead of one big ellipse, we see a few smaller ones. Near the start, the search spreads out a lot and wastes time in dead ends, but once it finds a way through, the ovals get longer and narrower as it heads toward the goal.



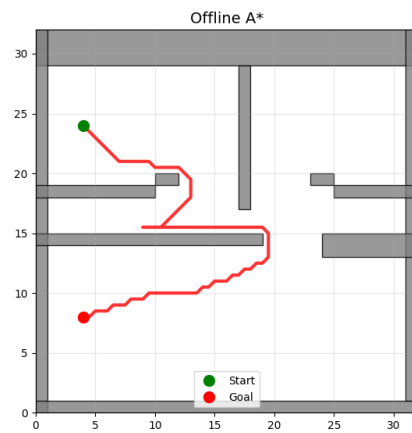
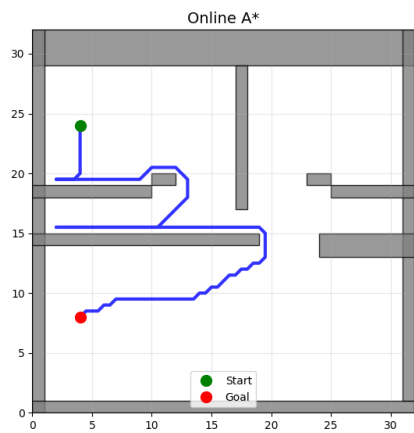
## Extra question:

In an online setting, the robot does not know the full map in advance. Instead, it senses obstacles and expands its knowledge of free space as it moves. At each step, the robot runs a localized A\* search on its current known grid, starting from its current position toward the target. Once the immediate next step is identified, the robot physically moves to that cell. It then scans again, updates the map with any newly detected obstacles, and repeats this process until the goal is reached.

Because replanning happens frequently and only with partial information, the online path is generally longer than the offline optimal path (which assumes full knowledge of the environment). The difference comes from detours taken when obstacles are discovered late in the navigation process.

We implemented the online navigation with a robot that has a sensor range of 3 cells. This means the robot can detect obstacles within a 3-cell radius of its current position. The robot starts knowing only the room boundaries and discovers interior obstacles (B1-B7) as it navigates.

Online: 68.8 | Offline: 47.7



### Results Comparison:

- Online path length: 68.8 units
- Offline optimal path length: 47.7 units
- Efficiency ratio: 1.44 (online path is 44% longer)